

Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Grafický editor s výstupem do HTML



2019

Zdeněk Mazurák

Vedoucí práce: RNDr. Arnošt Ve-
čerka

Studijní obor: Informatika, prezenční
forma

Bibliografické údaje

Autor:	Zdeněk Mazurák
Název práce:	Grafický editor s výstupem do HTML
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	RNDr. Arnošt Večerka
Počet stran:	31
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliographic info

Author:	Zdeněk Mazurák
Title:	Graphic editor with export to HTML
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Computer Science, full-time form
Supervisor:	RNDr. Arnošt Večerka
Page count:	31
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem této bakalářské práce je vytvoření grafického editoru pro Windows. Aplikace umožní uživateli kreslit a editovat grafické tvary a text. Všem objektům bude možno nastavit výplň či okraj barvou či gradientem. Editor bude umožňovat export do formátu HTML.

Synopsis

The aim of this bachelor's thesis is to create the graphic editor for Windows. The application will allow the user to draw and edit graphical shapes and text. The user will be able to change the color or gradient of both filling and frame of all present subjects. The editor will allow an exportation to HTML format.

Klíčová slova: grafický editor; grafické tvary; HTML CANVAS

Keywords: graphic editor; graphic shapes; HTML CANVAS

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce

podpis autora

Obsah

1	Úvod	8
2	Použité technologie	9
2.1	Programovací jazyk C#	9
2.2	WPF	9
2.3	XAML	9
3	Struktura aplikace	9
3.1	Grafické rozhraní	9
3.2	Třída MainControl	10
3.3	Třída FileControl	10
3.4	Třída DrawControl	10
3.5	Třída Layer	11
3.6	Třída Shape	12
3.6.1	Implementace vlastního grafického objektu	12
3.6.2	Editační prvky	14
3.6.2.1	Třída MovePoint	14
3.6.2.2	Třída EditRect	16
3.7	Třída HistoryControl	16
3.7.1	Metody třídy	16
3.7.2	IHistoryNode	17
3.7.2.1	Rozšíření IHistoryNodeSkipped	17
4	Funkce	18
4.1	Podporované soubory	18
4.1.1	Uložení do souboru ve formátu HTML	19
4.1.1.1	HTML soubor	19
4.1.1.2	Značka <canvas>	19
4.1.1.3	JSON	20
4.1.1.4	Vykreslení obrázku v prohlížeči	20
4.1.1.5	Datová struktura JSONu	20
4.1.1.6	Struktura uloženého souboru	21
4.1.2	Otevření souboru ve formátu HTML	21
4.1.3	Uložení do formátů rastrové grafiky	23
5	Uživatelské rozhraní	23
5.1	Horní panel	24
5.2	Levý panel	24
5.3	Kreslící plátno	25
	Závěr	27
	Conclusions	28

A Obsah přiloženého CD/DVD	29
Seznam zkratk	30
Literatura	31

Seznam obrázků

1	Diagram třídy Shape a jejích potomků	15
2	Diagram tříd implementující rozhraní IHistoryNode	18
3	Diagram datové struktury pro uložení obrázku do JSONu	22
4	Horní panel	24
5	Aplikace	25

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1	Nastavení automatické transformace revScale	14
2	Ukázka kódu HTML	19
3	Kreslení obdélníku do canvasu v JavaScriptu	19
4	Struktura HTML souboru	23

1 Úvod

Při tvorbě aplikace byl kladen důraz na jednoduché a uživatelsky přívětivé rozhraní. Hlavní inspirace pochází ze známého programu Malování z MS Windows od Microsoftu, který je jednoduchý, ale má řadu nevýhod, například nepodporuje průhlednost a hlavně všechno to, co je jednou nakresleno, už později nelze změnit. Oproti malování náš grafický editor tyto nevýhody nemá. Obsahuje navíc správu vrstev a u grafických objektů kromě barev lze použít i přechody barev (gradienty). Aplikace podporuje dva druhy přechodů: lineární a radiální.

Program není určený pro profesionály, ale pro lidi, kteří si za běžných okolností vystačí pouze s Malováním z MS Windows, ale občas se jim hodí nějaké pokročilé funkce. Pro většinu práce s obrázky si vystačíme pouze se základními funkcemi, proto není důležité mít přeplněný program plný zbytečných funkcí, které nám budou brát na přehlednosti.

Pro obyčejné smrtelníky ve světě grafiky je důležité, aby uživatelské rozhraní bylo jednoduché a přehledné. Toho chci docílit tak, že budu vycházet z jednoduchých principů kreslení.

2 Použité technologie

Následující kapitula popisuje jednotlivé technologie, které jsou v aplikaci použity.

2.1 Programovací jazyk C#

C# je programovací jazyk od společnosti Microsoft, který je hodně podobný jazyku Java a hodně čerpá syntax z jazyka C. Je objektově orientovaný a typovaný. Neexistují zde žádné globální proměnné a ani funkce, všechno musí ležet v nějaké třídě. Pokud je i přesto potřebujeme, lze využít statických tříd. Třídy kromě atributů a metod mají navíc vlastnosti. S využitím vlastností nemusíme neustále programovat gettery a settery, ale jsou automaticky vygenerovány za nás. [1]

2.2 WPF

Windows Presentation Foundation je knihovna pro tvorbu grafického rozhraní pro Windows od společnosti Microsoft. Je následníkem Windows Forms. Podporuje 2D, 3D grafiku a animace. K vykreslování dat může využít data binding. [2]

Data binding je technika, která má na starosti synchronizaci dat zobrazených s nějakým zdrojem dat. Existuje jednosměrný a obousměrný. Jednosměrný data binding aktualizuje zobrazovaná data v době, kdy nastane změna ve zdrojových datech. Obousměrný jej rozšiřuje tím, že aktualizuje zdroj dat z těch dat, která se zobrazují.

2.3 XAML

Extensible Application Markup Language je značkovací jazyk od Microsoftu. Pochází ze značkovacího jazyka XML. Původně byl vyvíjen pouze pro knihovnu WPF, nyní lze použít i v platformně Silverlight.

XAML nám umožňuje popsat uživatelské rozhraní jednodušším způsobem, než vytvářením objektů a nastavováním různých vlastností v programovacím jazyce. [3]

3 Struktura aplikace

Následující kapitola popisuje implementaci jednotlivých částí aplikace.

3.1 Grafické rozhraní

Grafické rozhraní aplikace (GUI) je vytvořené za pomoci knihovny WPF a z velké části popsané jazykem XAML. Jedna z mála věcí, která není popsána pomocí XAMLu je plátno, do kterého se kreslí a upravují se zde grafické objekty.

Hlavní okno je popsané v souboru `MainWindow.xaml` a obsluhu pro toto okno tvoří třída `MainWindow`. Z této třídy startuje celá aplikace a vyvábí se zde instance třídy `MainControl`. Všechny události jako jsou kliky myši na tlačítka nebo pohyby myši nad kreslícím plátnem jsou přeposílány třídě `MainControl`, která zase přeposílá zprávy nazpět třídě `MainWindow`, pokud je v okně potřeba něco změnit.

3.2 Třída `MainControl`

Třída `MainControl` má na starosti celý chod aplikace, udržuje v sobě právě zvolený nástroj, barvu a seznam otevřených souborů. Z okna aplikace si bere WPF element `Canvas`. `Canvas` je element, do kterého lze přidávat další elementy. Každý soubor má svůj `Canvas`, který je při práci na souboru vložen do hlavního `Canvasu`. Pokud se přepneme na jiný soubor, `Canvas` je vyměněn za `Canvas` jiného souboru. Je tu ještě další `Canvas`, který stojí úplně nad všemi ostatními. Tento `Canvas` slouží pro odchyťávání událostí a je také předán aktuálnímu souboru.

3.3 Třída `FileControl`

Třída `FileControl` reprezentuje otevřený soubor s obrázkem a stará se o práci s ním. Instancí této třídy můžeme mít v aplikaci teoreticky neomezené množství. Pokud je soubor aktivní, dostává všechny informace o vybraném nástroji a barvě. Dále je tento soubor informován o všech událostech, které se dějí v hlavním plátně - těmi jsou pohyby a kliky myši a z těchto pohybů je možno provádět kreslení a editaci. Kreslení a editace se delegují na třídu `DrawControl`.

`FileControl` uchovává seznam vrstev. Tento seznam je díky obousměrnému data bindingu automaticky synchronizován s panelem vrstev, který se nachází vlevo dole v grafickém rozhraní. Nese informaci o aktivní vrstvě. Aktivní vrstva je důležitá pro kreslení. Při začátku kreslení je grafickým objektům předána, aby objekt věděl, do jaké vrstvy se má nakreslit. Pokud se nacházíme v režimu editace, tak při změně aktivní vrstvy je informace o změně vrstvy předána grafickému objektu, který změní aktivní vrstvu na novou vybranou.

Pár věcí se změní pokud máme vybraný nástroj pro výběr grafického objektu. Aktuální vrstvě se předá zpráva o tom, že je aktivní a že má aktivovat všechny grafické objekty, které v ní leží. Následně všechny objekty ve vrstvě čekají na kliknutí. Při tomto kliknutí na objekt se opět nacházíme v režimu editace, kdy se nám zvolený objekt opět označí.

Každý soubor obsahuje svoji historii změn, o kterou se stará třída `HistoryControl`.

3.4 Třída `DrawControl`

Stará se o kreslení a editování objektů.

Může se nacházet v několika důležitých stavech. První z nich je stav, kdy chceme teprve něco nakreslit. Pokud v tomto stavu máme vybraný nástroj pro

kreslení tvarů, tak tato třída vytvoří jeho instanci a uloží si ji s bodem, kde se nacházela myš při začátku kreslení, kterou pak využívá pro případné rozpoznání, že uživatel nepohnul myší.

Další stavem je samotné kreslení, do kterého se přepne ihned po začátku kreslení. V tomto stavu přeposílá instanci grafického tvaru zprávy při každém pohybu myši a informuje ho o její pozici. Také dostává informace, jestli je provedeno kliknutí nebo odkliknutí myši. Jakmile kreslený objekt rozhodne, že je konec kreslení, přepne se `DrawControl` do stavu editace. O ukončení kreslení rozhoduje samotný grafický objekt kvůli tomu, že existují takové objekty jako například polygon, který se kreslí na více kliknutí myši, a tak nelze z jednoho kliknutí myši určit, zda má kreslení skončit. V tomto se liší například od úsečky, kde nám stačí pouze jeden tah myši. Pokud se jedná o objekt na jeden tah, tak se při odkliknutí ještě porovnávají souřadnice myši při odkliknutí se souřadnicemi uloženými na začátku kreslení, a když se rovnají, zruší se kreslení, odstraní se instance grafického objektu a třída `DrawControl` se přepne se do stavu na začátku kreslení.

Stav editace je stav, kdy je grafický objekt označený a určený k editování. V tomto stavu je grafický objekt připraven na editaci. Můžeme měnit jeho polohu a polohu jeho bodů. Můžeme také měnit barvy jeho výplně či čar anebo určovat tloušťku těchto čar.

Pokud na objekt klikneme, přepne se do mezistavu, který se nazývá přesouvání. Při přesouvání získáme od grafického objektu bod, je jedno jaký, jen je potřeba, aby grafický objekt věděl, o který bod jde a pamatoval si ho. Grafický objekt bude tento bod potřebovat při výpočtu souřadnic na které se má přesunout. Při pohybu myši třída `DrawControl` informuje grafický objekt o přesunu a společně s touto informací mu předá bod, který vznikne tak, že k tomu původnímu bodu přičte vektor posunu.

Při odkliknutí myši je přesouvání ukončeno a `DrawControl` se přepne nazpět do stavu editace. Označený objekt lze kromě přesouvání dále upravovat. Můžeme měnit jeho rozměry díky bodům, které grafický objekt zobrazuje.

Pokud klikneme mimo editovaný objekt, tak se `DrawControl` přepne nazpět do stavu na začátku kreslení a ihned můžeme kreslit nový objekt.

3.5 Třída `Layer`

Třída `Layer` reprezentuje vrstvu a uchovává v sobě její název, barvu pozadí, seznam grafických objektů, které obsahuje, informace o tom jestli je vrstva viditelná a také WPF element `Canvas`, do kterého přidává elementy z grafických objektů.

Třída implementuje rozhraní `INotifyPropertyChanged`, aby byl možný obousměrný data-binding. Informuje tak GUI o změnách, aby bylo jasné, že ve vrstvě nastala změna a je potřeba okno překreslit. GUI se pak překreslí podle dat z vrstvy.

3.6 Třída Shape

Tato třída reprezentuje grafický objekt a implementuje základní rutiny, které mají všechny grafické objekty totožné. Například barvy, změnu vrstvy, mazání nebo opětovné obnovení při akci zpět po smazání.

3.6.1 Implementace vlastního grafického objektu

Tato kapitola popisuje, jak implementovat do aplikace vlastní grafický objekt.

Pro vytvoření vlastního grafického tvaru je potřeba naprogramovat třídu. Třída musí dědit z třídy `Shape` a implementovat následující metody:

- `Konstruktor Shape(FileControl c, Layer la)`
- `Konstruktor Shape(FileControl c, Layer la, Serializer.Shape s)`

Je nutné zavolat původní konstruktor.

- **protected void** `OnDrawInit()`

Volá se v momentě, kdy objekt začínáme kreslit. V této metodě je nutné vyrobít instanci typu `System.Windows.UIElement` a uložit ji do vlastnosti `Shape.Element`.

- **protected void** `OnCreateInit(Serializer.Shape shape)`

Volá se v momentě, kdy je objekt načítán ze souboru. V této metodě dostaneme argument typu `Serializer.Shape`, který obsahuje všechny data pro sestavení tvaru (body, barvy atp.). Je nutné vyrobít instanci typu `System.Windows.UIElement` a uložit ji do vlastnosti `Shape.Element`.

- **protected bool** `OnChangeBrush(BrushEnum brushEnum, Brush brush)`

Tato metoda je volána pokud uživatel mění barvu objektu. V argumentu dostaneme informaci o jaký typ barvy se jedná a samotnou barvu. Pokud náš nový grafický objekt typ barvy podporuje a chceme akci o změně barvy uložit do historie, pak vrátíme hodnotu `true`, v opačné případě vrátíme hodnotu `false`.

- **protected bool** `OnChangeThickness(double thickness)`

Tato metoda je volána pokud uživatel mění u objektu tloušťku. V argumentu dostaneme novou tloušťku. Pokud náš nový grafický objekt tloušťku podporuje a chceme akci o změně tloušťky uložit do historie, pak vrátíme hodnotu `true`, v opačné případě vrátíme hodnotu `false`.

- **protected void** `OnDrawMouseDown(Point e, MouseButtonEventArgs ee)`

- **protected void** OnDrawMouseMove(Point e)
- **protected void** OnDrawMouseUp(Point e, MouseButtonEventArgs ee)

Předchozí metody se volají při kreslení. První při kliknutí dolů, druhá při pohybu myši a poslední při puštění tlačítka. Argument `Point e` obsahuje relativní souřadnice k plátnu a z argumentu `MouseButtonEventArgs` je možno získat informaci o myši (stisknuté tlačítka, souřadnice atp.) Ve chvíli kdy je grafický objekt nakreslen, zavoláme metodu `StopDraw()`, která ukončí proces kreslení a `SetActive()`, která aktivuje objekt pro editační mód.

- **protected void** CreateVirtualShape()

Slouží k vytvoření k průhledného WPF elementu. Je použit ke klikání na grafický objekt. Průhledný element musí mít nastavenou událost na kliknutí myši na metodu předka `Callback`.

- **protected void** SetActive()

Tato metoda je volána při aktivaci grafického elementu. Je nutné nastavit aktuálně použité barvy a tloušťku zavoláním `File.SetPrimaryColor(Brush c)`, `File.SetSecondaryColor(Brush c)` a `File.SetThickness(double t)`. Nemusíme použít všechny, záleží na tom, jestli náš grafický objekt, který programujeme, tyto vlastnosti používá. Dále je tu prostor pro zobrazení editačních prvků do horního canvasu.

- **protected void** OnMoveDrag(Point e)

Volá se pokud je grafický objekt aktivní při každém pohybu myši po plátně. Je možno využít k pohybu editačních prvků.

- **protected void** OnStopDrag()

Je zavolána při odkliknutí myši v editačním módu.

- **protected void** OnStopEdit()

Volá se při ukončení editačního módu.

- **public** Point GetPosition()

Návratovou hodnotou metody musí být bod. Při přesouvání objektu je tento bod využit a dá se pomocí něho spočítat, kam se má objekt přesunout.

- **protected void** OnMoveShape(Point point)

Volá se při přesouvání objektů. Jako argument dostává bod, který vrací metoda `GetPosition()` s přičtením vektoru posunu.

- **public** `Serializer.Shape CreateSerializer()`

Metoda musí vrátet potomka třídy `Serializer.Shape`. Slouží k serializaci objektu při ukládání do HTML souboru.

- `CreateImage(Canvas canvas)`

Musí se zde vytvořit nový WPF element a nelze použít element z vlastnosti `Element`. Tato metoda slouží ke skládání obrázku z toho důvodu, protože je volána v jiném vlákne, než ve kterém běží hlavní okno aplikace. WPF elementy vytvořené v jednom vlákne nelze používat v jiném.

Vytvořený element následně přidáme do `Canvasu`, který získáme z argumentu.

- **void** `ChangeZoom()`

Volá se v momentě, kdy je změněno měřítko zobrazení plátna. Z `File.RevScale` je možno získat poměr zvětšení tak, aby byl grafický objekt při změně přiblížení pořád stejně velký. `File.RevScale` se hodí pro zachování velikosti. Můžeme ho využít například pro zachování velikostí ovládacích bodů objektů. Samotný `File.RevScale` jde přímo vložit do vlastnosti WPF elementu tak, aby měnil velikost sám. Příklad zdrojový kód č. 1).

```
1 TransformGroup g = new TransformGroup();
2 g.Children.Add(revScale);
3 element.LayoutTransform = g;
```

Zdrojový kód 1: Nastavení automatické transformace `revScale`

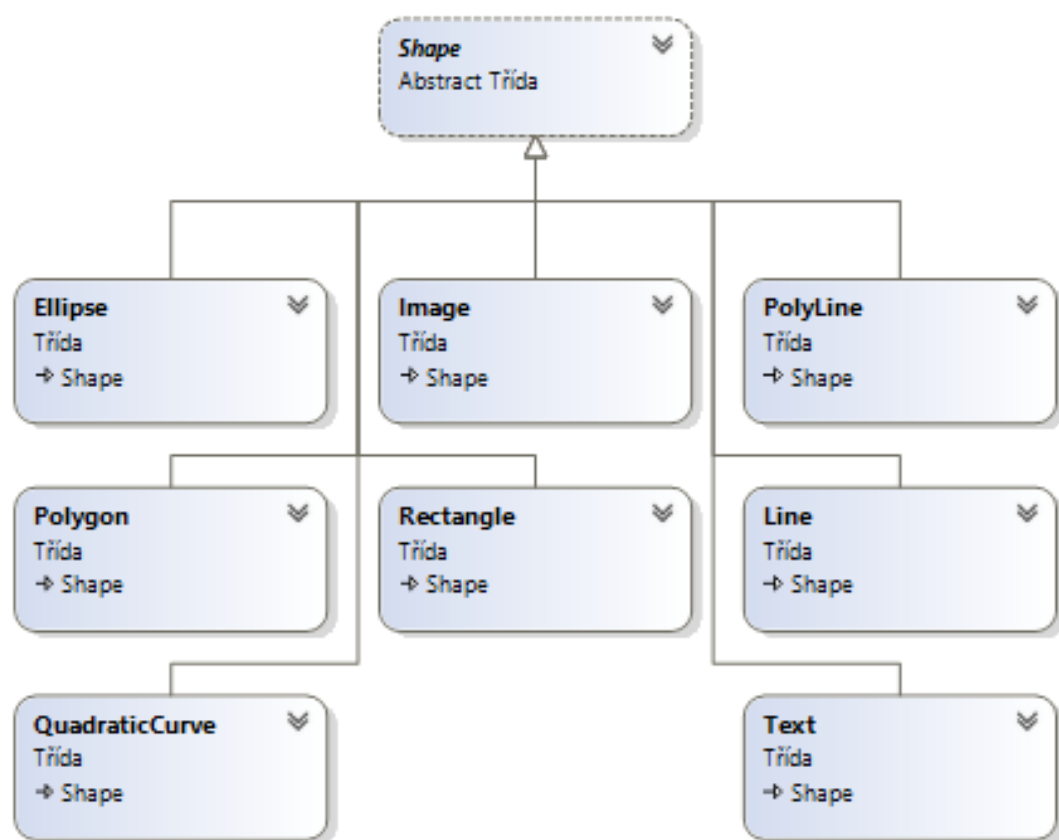
3.6.2 Editační prvky

Každý grafický objekt by měl mít k dispozici editační prvky, které se zobrazí v momentě, kdy je objekt editován. Editační prvky jsou WPF elementy s naprogramovanými událostmi, které vkládáme při editaci do vrchního `Canvasu`, aby je uživatel viděl před objekty a mohl nimi posouvat.

Jako příklad uvedu grafický objekt reprezentující úsečku, která při editaci přidává dva maličké kruhy (elipsy). Jelikož tyto elipsy jsou potřeba při editaci různých grafických objektů, vznikla proto speciálně určená třída.

3.6.2.1 Třída `MovePoint` Jak jsem již zmínil, pro ovládací prvky typu malých elips se kterým jde posouvat, vznikla speciální třída pod názvem `MovePoint`.

K založení instance je zapotřebí počátečního bodu, instanci grafického objektu, horního `Canvasu` a delegátu **void** `MoveDelegate(Point b)`. Delegát je funkce, kterou můžeme ukládat do proměnných a předávat ji jako argumenty



Obrázek 1: Diagram třídy Shape a jejích potomků

funkcí. Náš delegát se zavolá při každé změně pozice a v argumentu dostane bod místa, kam se má přesunout.

Pokud chceme editační prvek zobrazit, zavoláme metodu `Show()`. V opačném případě zavoláme metodu `Hide()`.

`MovePoint` automaticky ukládá všechny přesuny do historie, takže tuhle režii nemusíme v implementaci grafických objektů řešit.

3.6.2.2 Třída `EditRect` Slouží k editování obdélníkových tvarů (obdélník, elipsa, obrázek). Na čtyři krajní body používá třídu `MovePoint`. Pokud je aktivní, tak zobrazuje obdélník přerušovanou čarou. Při podržení klávesy `shift` a přesouvání jednoho z krajních bodů, drží poměr stran. Tato funkce se hodí například při zmenšování nebo zvětšování obrázků.

3.7 Třída `HistoryControl`

Každý soubor má svoji historii všech změn, které při práci v souboru provedeme. O tuto historii se stará třída `HistoryControl`. Každá třída `FileControl` si při vytvoření instance vytvoří svou vlastní instanci třídy `HistoryControl`.

`HistoryControl` v sobě uchovává dva zásobníky. První zásobník slouží k historii a jsou v něm uloženy akce, které byly provedeny. Do druhého zásobníku se přesouvají akce z prvního, pokud se v historii vracíme zpět. Když půjdeme vpřed, akce z druhého zásobníku se naopak přesune do prvního. Pokud provedeme nějakou změnu, zásobník určený pro kroky vpřed je vyprázdněn.

Jednotlivé akce jsou reprezentovány instancemi tříd implementujících rozhraní `IHistoryNode`.

3.7.1 Metody třídy

Třída `HistoryControl` má následující veřejné metody:

- **`void Enable()`**

Při vytvoření instance třídy `HistoryControl` je historie deaktivována. Zavoláním této metody spustíme celou funkcionalitu třídy.

- **`void Add(IHistoryNode node)`**

Tato metoda slouží k přidávání akce do historie.

- **`void Clear()`**

Smaže celou historii.

- **`void Back()`**
- **`void Forward()`**

Pohyb v historii vpřed a vzad, pokud tento přesun není možný, nic se neprovede.

- **void** SetNotChange()

Nastaví stav tak, jako by neproběhla vůbec žádná změna. Volá se při uložení souboru.

- **bool** Change()

Testuje, jestli se od posledního zavolání SetNotChange() nebo od vytvoření zapnutí historie tato historie nějak změnila. Používá se k testu, jestli je potřeba soubor uložit. Aplikace tento test používá k tomu, abychom si omylem nezavřeli aplikaci bez uložení.

3.7.2 IHistoryNode

IHistoryNode je rozhraní, které musí implementovat všechny třídy, které mají implementovat jednotlivé akce v historii.

Každá třída implementující toto rozhraní, musí implementovat následující dvě metody:

- **void** Back()

Zde je potřeba naprogramovat co se má stát, pokud chceme tuto akci vrátit zpět.

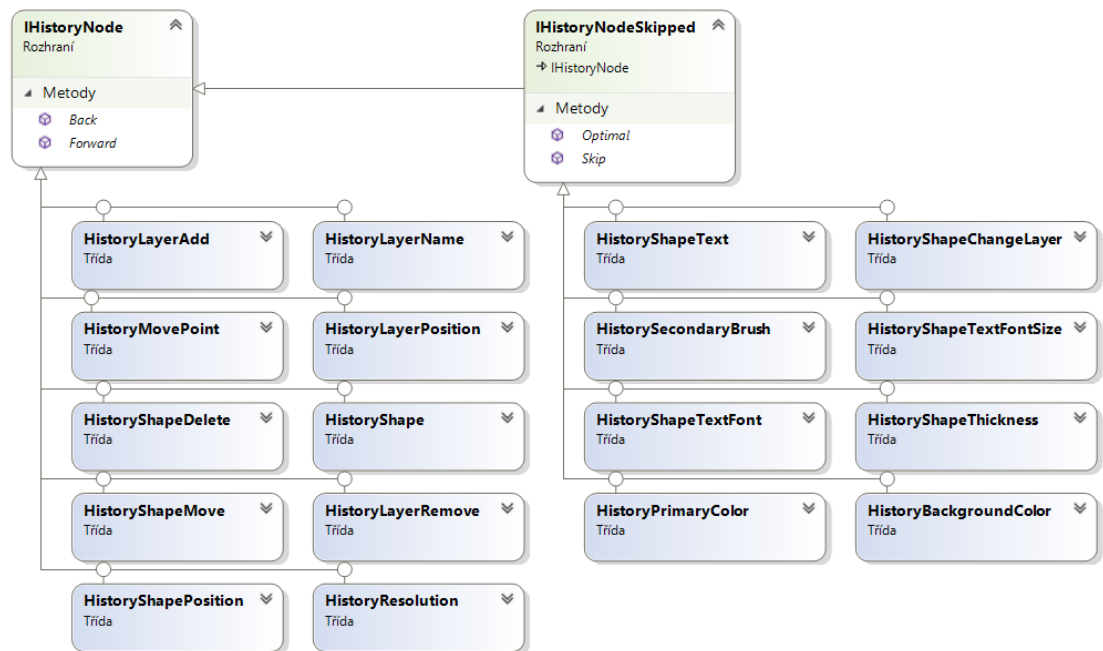
- **void** Back()

Opačný přístup, který se provede pokud byla akce již alespoň jednou v historii vrácena a má se zase vrátit do původního stavu.

3.7.2.1 Rozšíření IHistoryNodeSkipped Rozhraní IHistoryNodeSkipped je rozšíření rozhraní IHistoryNode. HistoryControl se však k němu chová odlišně.

Pokud metoda Add(IHistoryNode node) třídy HistoryControl dostane instanci třídy implementující rozhraní IHistoryNodeSkipped, tak provede následující kroky:

- Otestuje, jestli je zásobník určený pro akce zpět prázdný. Pokud je prázdný, postupuje klasickým způsobem.
- Pokud zásobník prázdný není, otestuje jestli akce ležící na vrcholu zásobníku neimplementuje rozhraní IHistoryNodeSkipped. Pokud ho neimplementuje, postupuje klasicky.
- Vkládaná akce s akcí ležící na vrcholu zásobníku jsou otestovány, jestli jsou kompatibilní. Pokud ano, akce se sloučí a vloží na vrchol zásobníku. V opačném případě se vloží vkládaná akce na vrchol zásobníku.



Obrázek 2: Diagram tříd implementující rozhraní IHistoryNode

Na test kompatibility dvou akcí IHistoryNodeSkipped slouží metoda `bool Optimal(IHistoryNodeSkipped node)`, která vrací pravdivostní hodnotu a určuje, zda jsou kompatibilní.

Sloučení dvou akcí provádí metoda `void Skip(IHistoryNodeSkipped node)`, kdy spojení probíhá do akce, na které je tato metoda volána.

Sloučení dobře slouží například u barev. Kdybychom měnili barvu objektu a ukládali bychom úplně každou změnu barvy, historie změn by byla nepoužitelná, protože by tam byl nespočet zbytečných změn, které by v práci překážely.

4 Funkce

4.1 Podporované soubory

Aplikace podporuje otevírání a ukládání souborů jako rastrovou grafiku ve formátech JPG, PNG a BMP. Při otevření těchto formátů je do plátna načten pouze jeden objekt a tím je celý obrázek. Nelze tak po opětovném otevření upravovat grafické tvary.

Speciální funkcí této aplikace je ukládání obrázků do formátu HTML. Oproti rastrovým formátům má výhodu v tom, že po otevření jsou načteny do plátna všechny objekty, které je možno dále upravovat. Tento formát reprezentuje obrázek jako vektorovou grafiku. Je možno ho otevřít v jakémkoliv prohlížeči, který podporuje HTML5 Canvas a má zapnutý JavaScript.

4.1.1 Uložení do souboru ve formátu HTML

4.1.1.1 HTML soubor Soubor s příponou .html nebo .htm je textový soubor, který obsahuje značkovací jazyk HTML. Jazyk se skládá z tagů (značek), které máme párové a nepárové. Do párových tagů můžeme vkládat další HTML tagy nebo text. Vzniká tak v HTML dokumentu stromová struktura. HTML tagy dále obsahují atributy, které mění vlastnosti tagů.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="content-type" content="text/html; charset=utf-8"
5       >
6     <title>Titulek</title>
7   </head>
8   <body>
9     <h1>Nadpis</h1>
10    <p>Text</p>
11  </body>
12 </html>
```

Zdrojový kód 2: Ukázka kódu HTML

4.1.1.2 Značka <canvas> Formát HTML není určený pro obrázky, ale pro webové stránky. Od páté verze HTML podporuje značku <canvas>, která reprezentuje plátno pro kreslení grafiky. Element se umístí do stránky s pevně danou šířkou a výškou. Do plátna pak lze kreslit přes API v JavaScriptu. Pokud změníme jeho rozměry, plátno se vymaže a musíme kreslit znovu.

```
1 var c=document.getElementById("myCanvas");
2 var ctx=c.getContext("2d");
3 ctx.rect(20,20,150,100);
4 ctx.stroke();
```

Zdrojový kód 3: Kreslení obdélníku do canvasu v JavaScriptu

Tato technologie nám nabízí pomocnou ruku, otázkou však zůstává, co uložit do HTML souboru, pokud požadujeme, aby se po otevření souboru ve webovém prohlížeči zobrazil obrázek.

Nabízí se možnost vygenerovat HTML soubor a vložit do něj vygenerovaný program v JavaScriptu, který dokáže obrázek do plátna nakreslit. Toto řešení by fungovalo dobře, ale mělo by řadu nevýhod. V HTML souboru by byla spousta nadbytečného kódu a implementace otevírání HTML souboru by byla příliš složitá. Nebylo by jednoduché dekódovat program z souboru na grafické objekty.

V budoucnu by také byl problém s kompatibilitou a to omezovalo by mě to v případné optimalizaci.

Grafické objekty v aplikaci jsou nějaká data uložená v objektech, a tak je jednodušší je tak nechat a zbytečně nepřidávat další zbytečný kód navíc. Pokud bych se vydal touto cestou stalo by se přesně to, co popisují v předchozím odstavci.

Druhou možností je uložit objekty do HTML souboru. Protože je HTML soubor textový, musíme tyto objekty reprezentovat pomocí textu. Chceme tedy tyto objekty převést na text a k tomu slouží serializace.

Serializace objektů je proces, který umožňuje objekty programu převést na posloupnost bytů nebo text. Jednoduše umožňuje objekty uložit pro pozdější obnovení.

4.1.1.3 JSON Aplikace používá serializaci do formátu JSON. JSON je datový formát, který dokáže pojmout datové struktury složené z čísel a řetězců. Často se používá k přenosům dat na internetu, hlavně ve webových aplikacích, které využívají technologii AJAX. Další obrovskou výhodou je, že pochází z JavaScriptu. Zkratka JSON totiž znamená „JavaScript Object Notation“ což je zápis objektů v JavaScriptu, a tak nemusíme programovat žádnou deserializaci, protože JavaScript deserializaci JSONu nativně podporuje, ale postačí nám pouze data v tomto formátu do souboru HTML přiložit. [6]

4.1.1.4 Vykreslení obrázku v prohlížeči V posledním kroku k zobrazení obrázku v prohlížeči zbývá vykreslit obrázek z přiložených dat v JSONu. Jak už jsem psal výše, použijeme k tomu HTML element `<canvas>`. Do canvasu se kreslí přes API v JavaScriptu, takže získaná přiložená data musíme dostat do programu v JavaScriptu, který bude dále obrázek vykreslovat do canvasu.

Data program získá deserializací JSON řetězce. Jelikož je JSON řetězec ekvivalentní se zápisem objektů v JavaScriptu, postačí nám pouze JSON řetězec přiložit do HTML souboru.

Program v JavaScriptu už jen přečte data a vykreslí všechny grafické objekty ve správném pořadí. Pro menší datovou velikost výsledného HTML souboru je kód programu zminifikován. Minifikace nám z kódu odstraní znaky, které nemají vliv na vykonávání kódu. Toto jsou například bílé znaky (mezery, tabulátory, řádky). Dále minifikace nahradí názvy proměnných za co nejkratší názvy. Po minifikaci program dělá to stejné jako před ní, jen je o hodně datově menší. Minifikace je hojně využívána na internetu a není se čemu divit, jelikož všichni chceme, aby se nám internetové stránky načítaly rychle.

4.1.1.5 Datová struktura JSONu Aby byl program v JavaScriptu schopný nakreslit obrázek z přiložených dat, musí mít data určitou datovou strukturu, která je velmi podobná struktuře, ve které aplikace uchovává data při práci s obrázkem.

Všechno začíná třídou `Picture`, která uchovává rozlišení obrázku a obsahuje vlastnost `layers`, ve které je uložen seznam vrstev. Jednotlivé vrstvy jsou reprezentovány třídou `Layer`, která v sobě uchovává barvu pozadí, jméno vrstvy a informaci zda-li je vrstva viditelná a jestli se má vykreslit. Mohli bychom říct, že je zbytečné přikládat vrstvu, která se stejně nenakreslí, ale jelikož jde HTML soubor v aplikaci znovu otevřít a plnohodnotně editovat, je dobré ji tam nechat, kdyby s ní ještě chtěl v budoucnu někdo pracovat. To nejdůležitější, co třída `Layer` obsahuje, je seznam grafických objektů umístěných ve vlastnosti `shapes`.

Grafické objekty jsou reprezentovány třídou `Shape`. Různé druhy grafických objektů mají hodně odlišné vlastnosti, a proto by nebylo moudré všechno ukládat, použijeme tedy dědičnost. Potomci třídy `Shape` jsou třídy reprezentující jednotlivé druhy grafických objektů, které v sobě uchovávají barvy výplní, okrajů a bodů, ze kterých jsou složeny nebo jiné vlastnosti.

Serializací objektů do formátu JSON přijdeme o informaci z jaké třídy pochází, jelikož JSON formát tyto informace neuchovává. Řešení je jednoduché, stačí přidat všem třídám reprezentujícím grafické objekty vlastnost `type`, kde každý druh bude mít svou unikátní hodnotu. V našem případě je vlastnost `type` typu řetězec a obsahuje vždy název své třídy. Díky tomu dokáže program jednoduchým způsobem rozpoznat o jaký druh grafického objektu se jedná a podle toho může očekávat vlastnosti tohoto objektu.

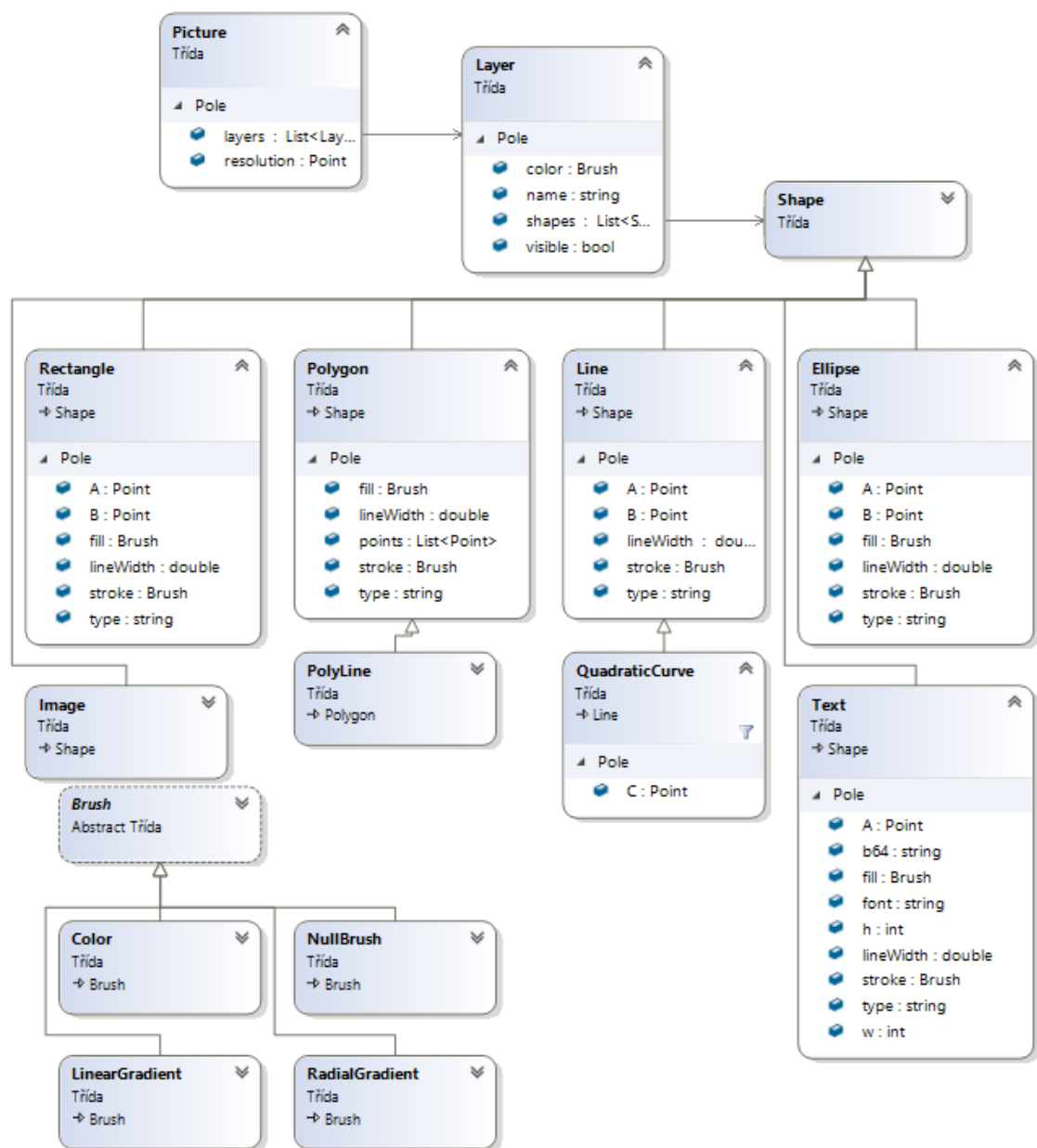
Zvláštním případem grafického objektu je obrázek, který reprezentuje třída `Image`, což je obrázek v rastrové grafice. Obrázek je uložen pomocí kódování `base64`. `Base64` je způsob kódování, který kóduje binární data do znaků ASCII, a tak lze tímto způsobem uložit obrázek pomocí řetězce. Umožní nám tak ukládat do HTML i rastrovou grafiku. Celou strukturu zobrazuje diagram na obrázku č. 3.

4.1.1.6 Struktura uloženého souboru Nyní máme všechny součásti pro uložení dat do souboru. Aplikace soubory HTML ukládá s následující strukturou (zdrojový kód č. 4).

4.1.2 Otevření souboru ve formátu HTML

Aplikace nám umožňuje otevřené HTML soubory plnohodnotně editovat. V souboru jsou zachovány všechny vrstvy a grafické objekty tak, jak byly uloženy. Toho jsme docílili díky tomu, že v JSONu jsou uloženy všechny informace a to i takové, které nemají vliv na grafickou podobu obrázku, což jsou například jména vrstev. Stejný přístup využívá aplikace Adobe Photoshop, kde můžeme obrázky ukládat do formátu PSD, který též zachovává všechny elementy.

Aplikace najde v HTML JSON řetězec a následně ho deserializuje. Z dat získaných z deserializovaného řetězce sestaví data pro soubor reprezentovaný v aplikaci a otevře ho v nové záložce.



Obrázek 3: Diagram datové struktury pro uložení obrázku do JSONu

```

1 <!DOCTYPE HTML>
2 <html>
3   <head>
4     <meta http-equiv="content-type" content="text/html; charset = utf
      -8">
5   </head>
6   <body>
7     <canvas width="500" height="400" style="border: 1px solid black;"
      id="MyPaint"></canvas>
8     <script>
9       var ctx = document.getElementById("MyPaint").getContext("2d");
10      var json = *JSON_RETEZEC*;
11      *PROGRAM*
12      draw(0, -1);
13    </script>
14  </body>
15 </html>

```

Zdrojový kód 4: Struktura HTML souboru

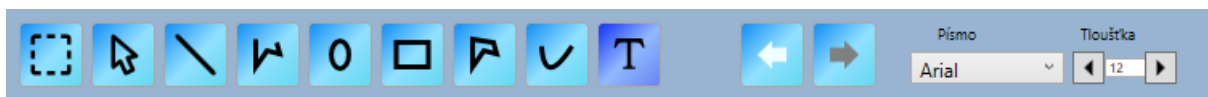
4.1.3 Uložení do formátů rastrové grafiky

Aplikace umožňuje uložit obrázky do rastrové (bitmapové) grafiky. Rastrová grafika je způsob ukládání obrázků. Obrázek tímto způsobem je uložen jako uspořádaná soustava bodů (pixelů), kde každý bod má svojí barvu. Barva pixelů je vyjádřena ve vybraném barevném modelu, ve kterém můžeme z posloupnosti čísel získat barvu, například model RGB, kde ze třech čísel reprezentujících 3 barvy (konkrétně červená, zelená a modrá) dostaneme výslednou barvou jejich smícháním. Existuje i rozšíření modelu RGB, což je model RGBa, kde je navíc ještě čtvrtá složka (alfa kanál), která určuje průhlednost barvy.

Implementace uložení do rastrové grafiky v Aplikaci je poměrně jednoduchá. Knihovna WPF umožňuje zachytit obrázek jakéhokoliv elementu, který se v GUI aplikace nachází. Aplikace ale nezachycuje přímo plátno, ve kterém probíhá práce na obrázku, ale pro urychlení vytvoří nové vlákno. V novém vlákně aplikace vytvoří nový element plátna a do a teprve něj přikládá všechny objekty, které mají být v obrázku vidět. Následně vyfotí plátno a výsledný obrázek už se pouze zakóduje do příslušného formátu a soubor s obrázkem se uloží na disk.

5 Uživatelské rozhraní

Následující kapitola popisuje rozložení prvků v uživatelském rozhraní a jejich funkce.



Obrázek 4: Horní panel

5.1 Horní panel

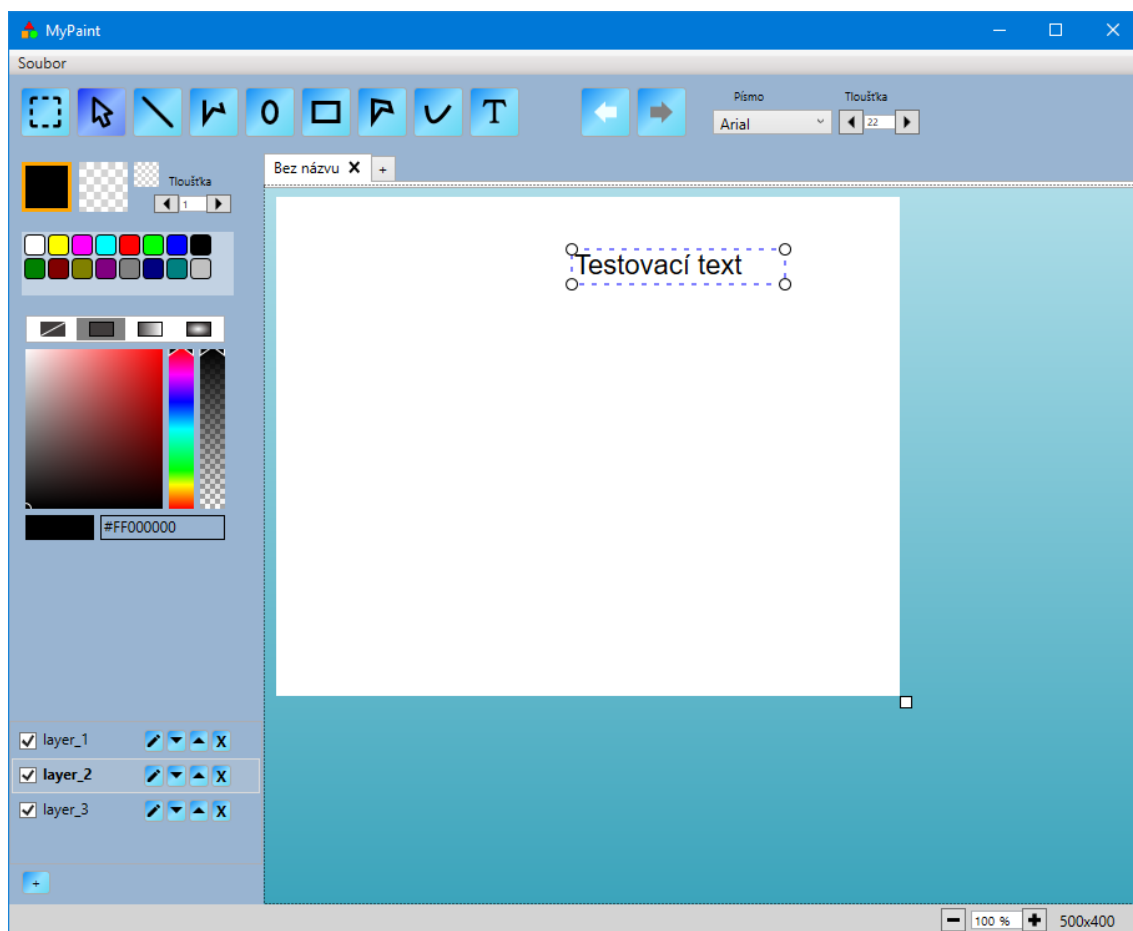
Horní panel aplikace slouží pro výběr nástroje pro práci. Funguje jako přepínač, současně lze zvolit pouze jeden nástroj. Následně uvedu jednotlivé nástroje:

- Nástroj pro výběr - slouží pro výběr části obrázku, který lze následně klávesovou zkratkou Ctrl+C zkopírovat do schránky a vložit jako výřez do jiného programu, nebo jej naklonovat.
- Nástroj pro výběr grafických objektů - slouží pro označení objektu, který chceme editovat.
- Úsečka - slouží pro kreslení obyčejné úsečky mezi dvěma body.
- Polyline - křivka vytvořená spojením n bodů.
- Elipsa
- Obdélník
- Polygon - slouží k vytvoření n-úhelníku.
- Kvadratická křivka - slouží k vytvoření Bézierovy křivky.
- Text - u textu je možné měnit velikost, font a barvu.
- Zpět - funkce zpět umožňuje vrátit se o libovolný počet kroků zpět.
- Vpřed - funkce vpřed je reverzní k funkci zpět.
- Nástroje pro změnu vlastnosti písma (zobrazí se pouze při editaci textu)

5.2 Levý panel

Horní část levého panelu slouží pro nastavení vlastností grafických objektů. Mezi hlavní vlastnosti patří barvy. V aplikaci můžeme každému objektu nastavit dvě různé barvy: primární a sekundární. Primární slouží u objektů jako barva okraje a sekundární jako barva výplně. Ne každý objekt může mít ale výplň. Například úsečka nebo kvadratická křivka výplň nemají, a tak na nastavení této vlastnosti nebudou reagovat.

Dále máme třetí barvu, která slouží pro nastavení pozadí aktuální vrstvy.



Obrázek 5: Aplikace

Pro změnu těchto barev přepínáme mezi tlačítka, které se nacházejí v levém horním rohu. Zvolíme si jednu z nich a následně jim nastavíme požadovanou barvu. Výběr je možno provést pomocí základních nadefinovaných barev umístěných níže. Pro pokročilejší výběr máme k dispozici paletu barev. Kromě jedné barvy lze nastavit i průhlednost nebo přechody více barev.

Na spodní části levého panelu nalezneme správu vrstev. Zde můžeme vrstvy přidávat a nebo mezi nimi přepínat, a tak měnit aktivní vrstvu, do které aktuálně chceme kreslit. Dále můžeme jednotlivé vrstvy mazat nebo měnit jejich jména či pořadí.

5.3 Kreslicí plátno

Ve středu aplikace se nachází kreslicí plátno. Zde můžeme vytvářet a následně upravovat grafické objekty. V pravém dolním rohu kreslicího plátna je umístěn bod pro změnu velikosti plátna. Při potažení za tento bod můžeme měnit libovolně velikost plátna. Nad plátnem se nachází záložky pro otevřené soubory, jelikož program umožňuje upravovat více obrázků najednou. Mezi těmito obrázky lze

přecházet klikáním na záložky.

Závěr

Cílem této bakalářské práce bylo vytvořit jednoduchý vektorový grafický editor, který umožňuje vytvářet a editovat jednoduché grafické tvary. Výsledný obrázek je možné uložit do formátu HTML, který když otevřeme v internetovém prohlížeči podporujícím HTML5, tak uvidíme daný obrázek. Dále nám tento editor umožňuje export do rastrových formátů jako je JPG, PNG nebo GIF. Při vytváření mě postupně napadaly další vychytávky, které bych normálně využíval, jako například: výřez z obrázku, přibližování, klávesové zkratky nebo vrstvy. Jelikož editor pracuje převážně s vektorovou grafikou, by bylo pěkné ho do budoucna rozšířit o export do formátu SVG. Při práci na aplikaci jsem se zdokonalil v jazyce C# a rozšířil si své znalosti hlavně v oblastech asynchronního programování a tvorbě GUI.

Conclusions

The aim of this bachelor thesis was to create a simple vector graphic editor which would allow us to generate and edit simple graphic shapes. The final picture can be saved as HTML and when we open it in the internet browser that supports HTML5, we can see that picture there. Furthermore this editor allows the export to raster formats as JPG, PNG or GIF. I came up with other ideas how to improve this editor which I would like to use while I was writing this code - for example: cut-out of picture, zooming, keyboard abbreviations or layers. Inasmuch as the editor works mainly with vector graphics, it would be nice to upgrade it with export into SVG. While creating this editor I improved in C# language and I have broadened my knowledge in asynchronised programming and GUI.

A Obsah přiloženého CD/DVD

bin/

Obsahuje adresář MICROSOFT .NET FRAMEWORK 4.5.2 ve kterém se nachází instalátor .NET Framework, který je potřeba pro spuštění aplikace.

V adresáři MYPAINT se nachází aplikace, která se spouští souborem MyPaint.exe

doc/

Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu, tj. zdrojový text textu, vložené obrázky, apod.

src/

Kompletní zdrojové kódy programu MYPAINT se všemi potřebnými knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu.

readme.txt

Instrukce pro spuštění programu MYPAINT, včetně všech požadavků pro jeho bezproblémový provoz.

Literatura

- [1] Wikipedia: Otevřená Encyklopedie: C Sharp [online]. 2019-07-29. [cit. 2019-07-29]. Dostupné z: https://cs.wikipedia.org/wiki/C_Sharp
- [2] Wikipedia: Otevřená Encyklopedie: Windows Presentation Foundation [online]. 2019-07-29. [cit. 2019-07-29]. Dostupné z: https://cs.wikipedia.org/wiki/Windows_Presentation_Foundation
- [3] Wikipedia: Otevřená Encyklopedie: Extensible Application Markup Language [online]. 2019-07-29. [cit. 2019-07-29]. Dostupné z: https://cs.wikipedia.org/wiki/Extensible_Application_Markup_Language
- [4] Wikipedia: Otevřená Encyklopedie: JavaScript Object Notation [online]. 2019-07-29. [cit. 2019-07-29]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript_Object_Notation
- [5] HTML5 Canvas - W3Schools [online]. 2019-07-29. [cit. 2019-07-29]. Dostupné z: https://www.w3schools.com/html/html5_canvas.asp
- [6] HTML5 Canvas - W3Schools [online]. 2019-07-29. [cit. 2019-07-29]. Dostupné z: https://www.w3schools.com/html/html5_canvas.asp